Appendix A

Explanations and Examples of construct of the ORM Grammar

<REMARK-SPEC> ::= REM [<any remarks>]

Explanation: Any line starting with REM is considered a remark (comment) line and it is ignored.

Example: REM This is a comment

<ORM-INFO> ::= [;ORMId=<ORMId>] [;ORMFile=<fileName>]

Explanation: <ORMId> specifies the Object Relational Mapping id of a specification. The default <ORMId> is the string "defaultORMId". The Object-Relational Mapping information (metadata) stored in the database corresponding to the given <ORMId> is used for subsequent processing.
An ORMFile specification overrides the mapping information corresponding to <ORMId>. This is an easy way to experiment with different mappings before storing that information permanently in the database.

Example: See <DATABASE-URL> below.

<DATABASE-URL> ::= <regularURL>[<ORM-INFO>]

Explanation: <DATABASE-URL> consists of the url (uniform resource locator, which includes database name, user name and password among other things) of the database to be connected to, optionally followed by ORM specific information <ORM_INFO>. <ORM_INFO> is used to initialize the database with the Object-Relational Mapping information or to retrieve the Object-Relational Mapping information from the database.

Example: See <DATABASE-SPEC> below.

<ENDDATABASE-SPEC> ::= ;

Explanation: This is just a delimiter to signify the end of <DATABASE-SPEC>

<DATABASE-SPEC> ::= DATABASE <DATABASE-URL> <ENDDATABASE-SPEC>

Explanation: A <DATABASE-SPEC> specifies the database and the Object-Relational Mapping (metadata) information to be used. Please see <DATABASE-URL> above for more details.

Example: DATABASE jdbc:odbc:sqlpubs; user=guest; password=hello; ORMId=pubs01;

or

```
          DATABASE jdbc:odbc:sqlpubs; user=guest; password=hello;
             ORMFile=pubs.jdx;
```

The first example specifies the use of Object-Relational Mapping information stored in the database corresponding to the ORMId "pubs01"

The second example specifies that the Object-Relational Mapping information should be retrieved from the file pubs. jdx

<PRIMARY-KEY-SPEC> ::= PRIMARY_KEY {<attribName> . . . }

Explanation: A <PRIMARY-KEY-SPEC> identifies the attribute names whose combined values uniquely identify a particular object. For a collection object, it specifies the attributes whose values are the same for all the objects in the collection.

Example: PRIMARY_KEY pub_id

or

PRIMARY_KEY title_id lorange

<REFERENCE-KEY-SPEC> ::= REFERENCE_KEY <referenceKeyName>
{<attribName> . . . }

Explanation: A <REFERENCE-KEY-SPEC> identifies the attribute names whose combined values uniquely identify a particular object. This may be an alternate way of identifying objects of a particular class. <REFERENCE-KEY-SPEC> is not allowed for collection classes.

Example: REFERENCE_KEY name fname minit lname

Here we are defining a reference key "name" consisting of three attributes - fname, minit and lname.

```
<SQLMAP-SPEC> ::= SQLMAP FOR <attribName>
                  [COLUMN_NAME <columnName>]
                  [SQLTYPE <sqlType>]
                  [NULLABLE]
```

Explanation: Through <SQLMAP-SPEC>, one can refine the mapping of a class attribute to SQL column in one of the following ways - use a column name different than the attribute name, use an SQL data type different than the default SQL data type for the attribute type, allow the column to be nullable. Allowing mapping of an attribute name to a different column name may be handy if the existing column name is cryptic and we want a more meaningful attribute name at the class definition level. Semantic knowledge of the data may be used to improve the storage efficiency for an attribute by specifying a more refined SQL type. For example, a String attribute (zipCode) may be mapped to varchar(10) instead of default varchar(255).

Some object-oriented languages like Java provide facility of reflection whereby the attribute names for a class and their types may be determined programmatically. If that is not the case, then a <SQLMAP-SPEC> needs to be specified for each attribute. Otherwise, some default mapping may be done using reflection facility.

Example: SQLMAP FOR prInfo COLUMN_NAME pr_info SQLTYPE text

or

SQLMAP FOR zip SQLTYPE varchar(10)

```
<RELATIONSHIP-SPEC> ::=  RELATIONSHIP <attribName>
                         REFERENCES <targetClassName>
                         {EMBEDDED | [BYVALUE] [REFERENCED_KEY
                         <referencedKeyName>]WITH<attribName>. . .}
```

Explanation: <RELATIONSHIP-SPEC> is used to provide details for a complex attribute.

EMBEDDED keyword means that the value of a complex attribute is embedded in a large binary column of the same table where rest of the primitive attributes are stored. This may be an optimized way for storing a referenced object if that referenced object does not need to be retrieved in any other context.

A Non-embedded complex attribute references a regular class or a collection class identified by <targetClassName>.

BYVALUE keyword implies that the referenced object (may be a collection object) does not have an independent existence without the existence of the containing object. When a containing object is stored, all the objects referenced through its BYVALUE complex attributes are also stored in the database. If a containing object is deleted, its BYVALUE referenced objects should also be deleted.

<referenceKeyName> specifies the name of a reference key of the class <targetClassName>. By default, referencing is done to the PrimaryKey of the target class.

The list of <attribName> is an ordered enumeration of the source attributes in the current class which are used to find the target class objects through the reference key. The data types of the source attributes should be compatible with the data types of the attributes defining the reference key in the target class.

Example:

```
RELATIONSHIP titles REFERENCES ArrayTitle BYVALUE WITH
pub_id
```

or

```
RELATIONSHIP job REFERENCES Job REFERENCED_KEY
PrimaryKey WITH job_id
```

The first specification means that the complex attribute 'titles' references an object of type ArrayTitle (which is a collection (array) of Title objects). The referenced object is contained in the current object by value. The attribute pub_id of the containing class is used to identify the (default primary key of the) referencing object.

The second example specifies that the complex attribute 'job' references an object of class 'Job' with the referencing object's attribute 'job_id' which should match the primary key attribute of the class 'Job'.

```
<ENDCLASS-SPEC> ::= ;
```

Explanation: This is just a delimiter to signify the end of a <CLASS-SPEC> or a <COLLECTION-CLASS-SPEC>.

```
<CLASS-SPEC> ::= CLASS<className>[TABLE<tableName>]<PRIMARY-KEY-SPEC>
                     [<REFERENCE-KEY-SPEC> .   .   .   ]
                     [<SQLMAP> .   .   .   ]
                     [<RELATIONSHIP-SPEC> .   .   .   ]
                     <ENDCLASS-SPEC>
```

Explanation: A <CLASS-SPEC> encapsulates all the Object-Relational Mapping information about one class. The <tableName> specifies the name of the relational table which holds the instances of this class. The default <tableName> is the same as the <className>. Other specifications have been explained earlier. Please note that it is mandatory to specify <PRIMARY-KEY-SPEC> for a class.

Example:

```
CLASS Title TABLE titles
PRIMARY_KEY title_id
RELATIONSHIP royscheds REFERENCES ArrayRoySched
BYVALUE WITH
   title_id
SQLMAP FOR price SQLTYPE Money
```

;

```
<ORDERBY-SPEC> ::= ORDERBY {<attribName> .   .   .   }
```

Explanation: An <ORDERBY-SPEC> of a <COLLECTION-CLASS-SPEC> specifies an ordered list of attributes whose values are used to sequence the objects in a collection during retrieval.

Example: ORDERBY ytd_sales title_id

The above specification for the collection class ArrayTitle means that such a collection of objects (e. g. in the titles attribute of a Publisher class object) should be ordered as per the values of ytd_sales and title_id attributes of the Title objects in the collection.

```
<COLLECTION-CLASS-SPEC> ::= COLLECTION_CLASS <className>
COLLECTION_TYPE {ARRAY | VECTOR}
     ELEMENT_CLASS <elementClassName>
                         [ELEMENT_TABLE <elementTableName>]
                         <PRIMARY-KEY-SPEC>
                         [<ORDERBY-SPEC>]
                         <ENDCLASS-SPEC>
```

Explanation: A <COLLECTION-CLASS-SPEC> encapsulates all the Object-Relational Mapping information about a collection class. A collection is actually a pseudo-class; there may not be an actual class by that name in the program.

The COLLECTION_TYPE specifies how the objects in the collection are combined together - in an array or in a vector.

The <elementClassName> specifies the class whose instances form the collection. Even the instances of a subclass of the <elementClassName> class may participate in a collection.

The mandatory <PRIMARY-KEY-SPEC> specifies the attributes which are the basis for realizing a collection. The values of these attributes are the same for all the objects in a collection.

The <elementTableName> specifies the name of the relational table which holds the instances of the collection objects. The default table is the same as the table for <elementClassName> class.

Other specifications have been explained earlier.

Example:

```
COLLECTION_CLASS ArrayRoySched COLLECTION_TYPE ARRAY
ELEMENT_CLASS
   RoySched
PRIMARY_KEY title_id
ORDERBY royalty
;
```

```
<ORM-SPEC> ::= <DATABASE-SPEC>
                  Any combination of <CLASS-SPEC>
                       <COLLECTION-CLASS-SPEC>,
                       <SEQUENCE-SPEC> and <REMARK-SPEC>
```

Explanation: An Object-Relational Mapping specification <ORM-SPEC> consists of <DATABASE-SPEC> followed by any combination of <CLASS-SPEC>, <COLLECTION-CLASS-SPEC> and <REMARK-SPEC>. This is what an <ORMFile> contains. The following example has an ORMId of pubs01.

This specification is contained in a file (pubs. jdx).

Example:

```
DATABASE
jdbc:odbc:sqlpubs;user=guest;password=hello;ORMId=pub
s01
;
REM
```

```
CLASS RoySched TABLE roysched
PRIMARY_KEY title_id lorange
;
COLLECTION_CLASS ArrayRoySched COLLECTION_TYPE ARRAY
   ELEMENT_CLASS RoySched
PRIMARY_KEY title_id
ORDERBY royalty
;
CLASS Title TABLE titles
PRIMARY_KEY title_id
RELATIONSHIP royscheds REFERENCES ArrayRoySched
BYVALUE WITH
   title_id
SQLMAP FOR price SQLTYPE Money
;
COLLECTION_CLASS ArrayTitle COLLECTION_TYPE ARRAY
ELEMENT_CLASS
   Title
PRIMARY_KEY pub_id
ORDERBY ytd_sales title_id
;
CLASS PubInfo TABLE pub_info
PRIMARY_KEY pub_id
```

```
SQLMAP FOR logo SQLTYPE image
SQLMAP FOR prInfo COLUMN_NAME pr_info SQLTYPE text
;
CLASS Publisher TABLE publishers
PRIMARY_KEY pub_id
RELATIONSHIP pubInfo REFERENCES PubInfo BYVALUE WITH
pub_id
RELATIONSHIP titles REFERENCES ArrayTitle BYVALUE
WITH pub_id
;
CLASS Job TABLE jobs
PRIMARY_KEY job_id
;
CLASS Emp TABLE employee
PRIMARY_KEY emp_id
RELATIONSHIP job REFERENCES Job REFERENCED_KEY
PrimaryKey WITH
   job_id
RELATIONSHIP publisher REFERENCES Publisher WITH
pub_id
;
CLASS TitlePub
PRIMARY_KEY title_id
```

```
            ;
            CLASS LinkList TABLE linklist
            PRIMARY_KEY link_id
            RELATIONSHIP next REFERENCES LinkList BYVALUE WITH
            next_link_id
            ;


<SEQUENCE-SPEC> ::= SEQUENCE <sequenceName>
                          MAX_INCREMENT <maxIncrementValue>
                          [START_WITH <startingVal>]
```

Explanation: A <SEQUENCE-SPEC> defines a sequencer which can provide chunks of persistently unique sequence numbers.

<maxIncrementValue> is used to do sanity-check against requests which may erroneously ask for a large chunk of sequences which may quickly reduce the availability of new sequence numbers.

Optional <startVal> specifies the starting sequence number provided through this sequencer. The default is 1.

Example: SEQUENCE seqFoo MAX_INCREMENT 100 or

```
SEQUENCE seqBar MAX_INCREMENT 1000 START_WITH 10001
```

The second sequencer (seqBar) starts with a value of 10001.